Naval Research Laboratory

Washington, DC 20375-5000

NRL Memorandum Report 6778

BaRT Manual
Version 3.0

NAVEEN HOTA, CONNIE LOGGIA RAMSEY, LI WU CHANG
AND LASHON B. BOOKER

Navy Center for Applied Research on Artificial Intelligence
Information Technology Division

February 14, 1991

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	1991 February 14	

4. TITLE AND SUBTITLE

BaRT MANUAL
Version 3.0

5. FUNDING NUMBERS

55-0230-0-1

PE 62234N

TASK AREA

RS34-C74-000

6. AUTHOR(S)

Naveen Hota, Connie Loggia Ramsey
LiWu Chang, Lashon B. Booker

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
ATTN: Code 5510
4555 Overlook Avenue
Washington, DC 20375-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL Memorandum
Report 6778

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

Unlimited

13. ABSTRACT *(Maximum 200 words)*

BaRT is an inference engine which as been developed to aid in classification problem solving. This inference engine uses Bayesian reasoning and can handle problems associated with incomplete and uncertain evidence. It has successfully been used to perform ship classification. This manual describes how to load the BaRT program and how to use all of the available commands. This manual also provides some theoretical background and some implementation details concerning BaRT.

14. SUBJECT TERMS

belief networks, Bayes probability, belief classification influence diagram, diognostic support, belief commitment probabilistic reasoning, causal support, knowledge acquisition

15. NUMBER OF PAGES

48

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR

CONTENTS

BaRT MANUAL
VERSION 3.0

1. Introduction

Many real world problems are associated with uncertainty; the evidence people observe which helps them to reason about some goal event is almost always uncertain and incomplete. Still, people make judgments based on this uncertain and incomplete evidence. These uncertain evidences can be combined in various ways to find the validity or strength of a hypothesis [6,8], and Bayesian probability theory is a *normative* theory that allows one to reason about and combine uncertainties. Pearl[9] and others have devised ways to represent, reason about and combine uncertain evidences in a way that conforms to the tenets of probability theory, but avoids the disadvantages usually associated with probabilistic computations of belief. BaRT is a Bayesian Reasoning Tool which implements some of these ideas. It has been implemented as an AI programming environment and it has been used in a prototype decision aid to classify ship images[1]. In BaRT, a problem is represented as a collection of networks called a domain model. The domain model can have any number of related knowledge structures (networks). The belief in each value of each hypothesis in a network can change as new evidence lends support to (or takes support away from) certain values of the hypothesis.

The rest of this manual is organized as follows. Section 2 provides an overview of the theoretical background for this work, and Section 3 provides some of the implementation details. Section 4 explains how to use BaRT and Section 5 explains how to use the knowledge acquisition module. Section 6 provides an example.

2. Knowledge Representation and Reasoning Schemes in BaRT

2.1. Belief Networks

Belief networks provide a method for hierarchical probabilistic reasoning in directed, acyclic graphs. Each node in the network represents a discrete-valued hypothesis which describes an aspect of the domain, and each node contains information about both the current belief of each value of the hypothesis and the most

Manuscript approved November 30, 1991.

probable instantiation of the hypothesis given the evidence available, called the belief* distribution. (The belief* distribution provides a way to determine how the various degrees of belief in hypotheses can be interpreted. Generating a coherent explanation involves the simultaneous acceptance of a set of hypotheses, a requirement that goes beyond simply noting the degree of belief in any individual hypothesis. This means that the problem solver must make a *commitment* in categorical terms about the best way to instantiate each hypothesis variable based on the evidence available.) Each link between two nodes represents a direct causal dependence between two of the hypotheses, and the directionality of the link is from cause to manifestation. Each node contains a tensor[1] of probabilities conditioned on the states of the causal variables; this tensor quantifies the relationship between a node and its parents (causes). It is important to note that numbers used to quantify the relationship do not have to be probabilities. All that is required is that the tensor entries are correct relative to each other.

The belief updating scheme used in BaRT is based on Pearl's message passing algorithm[6] for singly connected networks. It keeps track of two sources of support for belief at each node: the diagnostic support derived from the evidence gathered by descendants of the node and the causal support derived from evidence gathered by parents of the node. Diagnostic support (λ) provides the kind of information summarized in a likelihood ratio for binary variables. Causal support (π) is the analogue of a prior probability, summarizing the background knowledge lending support to a belief. These two kinds of support are combined to compute the belief at a node with a computation that generalizes the odds/likelihood version of Bayes' rule. Each source of support is summarized by a separate local parameter, which makes it possible to perform diagnostic and causal inferences at the same time. These two local parameters (λ and π), together with the tensor of numbers quantifying the relationship between the node and its parents, are all that is required to update beliefs. Incoming evidence perturb one or both of the support parameters for a node. This serves as an activation signal, causing belief at that node to be recomputed and support for neighboring nodes to be revised. The revised support is transmitted to the neighboring nodes, thereby propagating the impact of the evidence. Propagation continues until the network reaches equilibrium. The overall computation assigns a belief to each node that is consistent with probability theory. Using a similar computation, similar supporting factors (π^* and λ^*) are used to find the belief* distribution. The equations for belief and belief* updating are presented in Appendix A. The reader is referred to Pearl [9] for more details about the theoretical framework of this belief maintenance system. BaRT converts networks with loops into singly connected networks using Chang and Fung's[10] node aggregation algorithm. Several auxiliary computations are also

1 A *tensor* is a mathematical object that is a generalization of a vector to higher orders. The order of a tensor is the number of indices needed to specify an element. A vector is therefore a tensor of order one and a matrix is a tensor of order two.

available in BaRT: Auxiliary variables to compute Boolean constraints and queries, and an error-based measure of impact to gauge the potential effect an uninstantiated node might have on the uncertainty of the target hypothesis [9].

2.2. Taxonomic Hierarchies

A taxonomic hierarchy represents set-subset relationships; each child of a given node represents a subset of that node. The leaf nodes of the hierarchy are a finite set

$$H = \{h1, h2, h3, ...\}$$

of mutually exclusive and exhaustive constituent elements. Certain subsets of H can be grouped to form a strict hierarchy in which each subset has one parent. The full set H is then the root node of this tree, and intermediate nodes represent the disjunction of their child nodes. Each node represents both a subset of H, and a hypothesis that is true whenever exactly one of its constituent elements is true.

Several hierarchies, based on the same set H of constituents, can be used at the same time in BaRT. These may have common leaf nodes, and the beliefs of all nodes will be updated when evidence is entered into any of the networks. Belief updating in taxonomic hierarchies is accomplished using Pearl's weight distribution/normalization procedure. See [9] for more information on taxonomic hierarchies and belief updating within them.

2.3. Influence Diagrams

An influence diagram is a belief network generalized to solve decision problems(Cooper[3]). In addition to the regular chance nodes, an influence diagram also has decision nodes and a value node. One of the goals in using an influence diagram is to determine the decision alternatives that maximize expected value. Only one value node is allowed in the network. BaRT solves influence diagram problems by using Cooper's method[3] to convert the influence diagram into a belief network. Cooper's algorithm recursively constructs and evaluates all paths in the decision tree that corresponds to the influence diagram. The efficiency of BaRT implementation is enhanced with a simple branch and bound technique. Before any path is explored, BaRT makes an optimistic assumption about the expected value that will be computed for that path. Whenever this optimistic bound is worse than the expected value of some fully expanded alternative, the candidate path is pruned.

3. Implementation

BaRT is implemented in Common LISP and CLOS (Common Lisp Object System) and comes with a window interface on Symbolics and Sun work stations. Presently it runs on Symbolics Genera 7.2, Lucid 2.1.3 and Lucid 3.0.

All of the core functions to run BaRT are defined in a package called *bart* which is in the file named *bart.lisp*. This does not use any window function and can be run on any machine with Common LISP. The functions in this file can be called from LISP using the conventions described in Appendix C. A non-graphic user interface is developed for use on any machine with Common LISP (see Appendix B); this code is in the file *bart-frame-tty.lisp*. The graphic user interface (machine dependent) is developed for two machines: Symbolics and Suns. The interface code for the Symbolics is in the file *bart-frame-3600.lisp* and for the Sun it is in *bart-frame-sun.lisp*. The name of the package containing the user interface functions is *bart-frame*. The object definitions used in the BaRT routines are defined in the file *bart-defs.lisp*. All the general utilities are in files *bart-util.lisp*, *sym-util-all.lisp*, and *sun-util-all.lisp*. *Bart-loops.lisp* contains code to convert networks with loops into singly connected networks. *Bart-rw.lisp* contains code to write the loaded networks into an internal file *fff-bart-int* where fff is the name of the input file.

The knowledge acquisition core functions are defined in a package called *bart-ka* and reside in the file named *bart-ka.lisp*. These functions must be used with the graphic interface on the Symbolics and Suns, and the graphic user interface code is in the files *bart-frame-3600-ka.lisp* and *bart-frame-sun-ka.lisp*. General utilities are in the files *bart-util.lisp*, *sym-util-all.lisp* and *sun-util-all.lisp*.

All the package definitions are in the file *pkgdefs.lisp*. The whole system is defined in the file *bart-defsys.lisp* which must be loaded first.

3.1. General Conventions

There are several conventions when using the BaRT window interface. To choose a command, the user should click-left on the command button. Clicking middle on the command button will provide a brief documentation in a pop-up window. After reading the documentation the user should press any character to get rid of that window.

When one of several command options needs to be selected, a pop-up menu appears with the options and the user has to click on one of these presented options with the mouse. Moving the mouse away from the menu aborts the selection.

An input field for a particular piece of information is a line in a window beginning with the name of the input item followed by a mouse sensitive field that can be edited. All these fields are initially filled with some default values. To edit any value, the user should click left on the mouse sensitive field. To enter a new value, the user should remove the present value by killing whatever is there in this field (see edit commands below) and enter a new value. The input mode for that field can be ended by a carriage return <CR> (no matter where the mark is). Each field has an internal buffer and only part of this buffer is visible in the field. Once the user gets to the end/beginning of the visible field, then the visible field moves to the next/previous portion of the current visible field of the buffer. A few editor commands are provided in the input mode for the user to edit this field. They are as follows:

<Cntl>-a -- move to the beginning of the visible field.

<Cntl>-e -- move to the end of the visible field.

<Cntl>-p -- previous visible field of the buffer.

<Cntl>-n -- next visible field of the buffer.

<Cntl>-k -- kill from point to end of buffer.

<Cntl>-y -- yank from kill ring at point.

<Cntl>-l -- redisplay the field.

Backspace -- remove a char before the mark.

Rubout -- remove the char at the mark.

Besides *Return*, *space* can also be used to terminate input mode where it is enough to recognize the end of input mode. For example, while typing just a number, space can be user to terminate input, as spaces are not allowed inside numbers.

Some input items allow the user to select a value from an existing set of choices. For a field expecting a boolean value, clicking on the field toggles the value of the field. Likewise when the system is expecting the user to choose only one value among a set of exclusive choices, it presents a box and a value for each choice. Selecting any one value in them would deselect the other values. If the system is expecting a non exclusive choice, the user can select as many choices from the presented set as he wishes by clicking on each box of his choice. In all the above cases, the box in front of the selected choice is filled to indicate that choice has been selected.

Whenever the system presents some temporary window with input fields, it also presents two buttons: *Done*, and *Abort*. Pressing on *Abort* closes the window and none of the changes made in that pane would take effect. Pressing on *Done* takes the new values and closes that window. Associated with each field are a type and optionally a function to check the validity of the input. Upon completion of the input, the system first checks and sees if the user has given the correct input type. Then it would call the function associated with this field with the current input as argument. If the given input is not of the proper type or not valid, then an appropriate error message would appear in the *messages pane* at the bottom of the window and lets the user correct the input.

Sometimes a prompt may appear in the messages pane to get input from the user. Usually this happens when the system needs to take only one value. In that case the user should type the new value in the messages pane in front of the prompt. Example: when the system needs to get a file name, it would prompt the user in the messages pane and then reads a new pathname.

The user needs to enter an expression while creating a query, constraint node and while filling the joint conditional probability matrices in the knowledge acquisition mode. Here the system reads the input, character by character, and parses the expression. This parsing has limited capabilities. Basically the expression can contain only four tokens along with the node names and their values. The four tokens are *and, or, not,* and *equal.* These can be substituted by the mathematical symbols &, |, ^, and = respectively. The = token along with the node names and their values is used to construct simple expressions. A simple expression is a list of exactly three elements. The first element is the sign =. The second element is the name of a valid node. The third element is the value name of the node name specified as the second element. An example of a simple expression is (= Node-a Value-of-A). The boolean operators *and, or,* and *not* can be used to construct compound expressions combining simple expressions. An example of a compound expression is *(& (| (= Nd-A Val2) (= Nd-C Val1)) (= Nd-B Val5)).* While constructing query and constraint nodes, any node in the current network is a valid node. The value name must be one of the values that the node was given when it was defined for the network. All the node and value names are case sensitive. The user should type the name as is and should not include double quotes around names. When constructing matrices in knowledge acquisition mode, valid nodes are only the current node and its parents. Here the user may give more than one expression to fill the matrices. So the user can supply a number between 0 and 1.0 after each expression. The system takes all the expressions and makes a cond statement with the expression part as the test and the number given as the number to be used in filling the entries in the conditional probabilities. If no number is supplied then it defaults to 1.0. After filling some or all entries in the conditional probability matrix, the system would then normalize the values in each column. The user should not supply this number while constructing query and constraint nodes and the number is always taken as 1.0.

The mouse cursor is normally a north-west pointing arrow. This changes to a circle with cross hairs in it when the system is expecting the user to choose a node or a location.

Note

Sometimes the mouse cursor on the SUN gets corrupted and won't return to the normal north-west point arrow. In this case call *(sun-util::reset-mouse-cursor)* from the lisp interaction pane to reset the mouse cursor to its normal shape.

The input the user gives is always case sensitive.

4. Using BaRT

A graphic interface has been developed for Symbolics and Suns. (In order to run the program without the graphic interface, the reader should see Appendix B.) To run BaRT, the user should load the system and then choose the appropriate commands. This is explained in detail below.

4.1. Compiling and Loading the BaRT module.

Symbolics: Get into a Common LISP environment with or without PCL. Edit the file *bart-defsys.lisp* to indicate the directory where BaRT files reside. From the LISP listener, load the file *bart-defsys.lisp*. If you just want to load the system then type *(load-bart)*. In this case it loads the appropriate source or binary files of the system. While loading these files, the system may load a file *CLOS* depending on whether PCL is in the environment. Now, invoke the program by first pressing the *Select* key and then pressing the *Symbol, Shift* and *B* keys simultaneously.

Sun: Edit the file *bart-defsys.lisp* to indicate the directory where BaRT files reside. Then invoke Common LISP with or without PCL in a sunview window and load the file *bart-defsys.lisp*. If you just want to load the system then type *(load-bart)*. In this case it loads the appropriate source or binary files of the system. While loading these files, the system may load a file *Clos* depending on whether PCL is in the environment. Change to package *bart-frame* by typing *(in-package 'bart-frame)*. Now, invoke the window environment by typing *(start-window)*. This creates a lisp pane in an editor environment. Then invoke the program by typing *(start-bart)* in that lisp pane.

To compile the system type *(load-bart t)*. This compiles and loads the system after compiling. It is strongly suggested that a disksave image be created for the Sun version. To do this after the files have been compiled and loaded, type the command *(disksave "Bart" :restart-function #'bart-frame::start-all :full-gc t)* before you invoke the window environment.

4.2. BaRT Windows and Commands

BaRT Windows

After loading the BaRT system, the whole screen consists of eight windows: the title pane, the belief network display pane, the constraint and query nodes pane, the global system parameters pane, the command menu pane, the node/link information display

pane, the LISP interaction window, and the message window. The belief network display pane, the constraint and query nodes pane, and the node/link information display pane are all scrollable in the standard way for the machine. Figure 1 provides a sample screen display.

The *title window* lies across the top of the screen and consists of the heading *Bayesian Reasoning Tool (BaRT)* in boldface.

The *belief network display pane* is on the left hand side of the screen occupying a large portion of the screen. This pane is used to display the networks of the current domain model. Some of the nodes in the network may be grayed (depending on whether the option to *compute measure of impact* is selected and whether a targetnode has been selected for that network); the intensity of the grayness measures the entropy (uncertainty) in the belief values of each node in the network as it relates to the target node (i.e., how much influence changing the beliefs in that node will affect the beliefs in the target node.)

The *query and constraint display pane* is on the top right hand side of the screen. This is where the query and constraint nodes appear if any.

The *global system parameters pane* is right below the query and constraint display pane and consists of two lines. The slot on the first line contains the current domain model (this will be empty before loading the data file). The slot on the second line contains the current knowledge structure. If the current knowledge structure name is in boldface, then that network is in equilibrium; otherwise the network is not in equilibrium. This distinction is useful when propagating the effect of new evidence in the network in step mode, i.e., updating one node at a time.

The *command menu pane* is right below the global system parameters pane and consists of the commands which are mouse sensitive and can be invoked by mouse-clicking left on them. Mouse clicking right on these provides a brief documentation of that command. All mouse-sensitive commands are highlighted when selected.

The *node/link information display pane* is below the command menu pane and is used to present information about nodes/links in the network.

In the bottom of the screen is the *interaction window* for normal interaction. Lisp expressions can be evaluated in this pane on the SUN. On the Symbolics, the user should press the *SUSPEND* key to get a top level lisp read-eval loop in this pane to try any lisp expressions. During this time, none of the commands are active. The user should press the *CONTINUE* key to get back from the suspend mode.

The message window is at the bottom of the screen and is used for displaying helpful messages and also for taking some inputs such as a file name.

Top-Level BaRT Commands

There are several conventions when using the commands in BaRT. Please refer to Section 3.1 *(General Conventions)*. Unless otherwise specified, all the commands are executed in the context of the current knowledge structure.

The possible command choices are:

Active-Structure :

Allows the user to change the current knowledge structure to a different one within the current domain model. A pop-up menu appears with the choices of all of the knowledge structures within this domain model. (If there is only one knowledge structure in the current domain model, this command will not do anything.)

Constraint :

Allows the user to add a constraint to the network by providing a boolean expression through a temporary menu. The boolean expression provided is set to true and the belief distributions of the other nodes in the network are adjusted. (It constrains the beliefs in the network to make the boolean expression true.) An example boolean expression is:

(or (equal node-1 True) (equal node-5 False))

Note that the node names and the values are always case sensitive. Please refer to Section 3.1 *(General Conventions)* for syntax. The effects of the new constraint can be propagated through the network by clicking on *Update*. The user can also choose to delete, disable or enable a constraint node. Disabling the node leaves it in the system, but the effects are taken away, as if it did not exist. Enabling the node brings the effects back again. Any of these operations would bring the network out of equilibrium, so the user should click on *Update* to bring the network into equilibrium again. Note that the name of the constraint node is in boldface type when active and not in boldface type when not active.

Default-modes :

Allows the user to set available options. After clicking on this, a temporary menu entitled *Select Default Modes* of all the user settable options appears. The user can change any of these values by clicking left on them. The global options appear with their present values in boldface. This command can be terminated by clicking either on *done* to process the request or *abort* to ignore the request.

Step mode shows the propagation in steps, i.e., the system propagates one node at a time. This is useful if the user wants to see the results after each update of a

node. Note that the user can tell whether the system has reached equilibrium; if the current knowledge structure name in the global system parameters pane is in boldface, then the network is in equilibrium.

*Update belief** determines whether the belief* vector of each hypothesis is updated when the system propagates the effect of new evidence.

Update lambda/pi determines whether the lambda and pis on each link is updated. This always happens when the network doesn't have loops. This option is used only for networks with loops; the lambdas and pis computed are approximate values and the computation is not guaranteed to converge.

Compute measure of impact computes a factor at each node *A* which is a measure of the uncertainty that can be reduced in the target node by adding evidence to node *A*. BaRT computes a relevance-based measure described by Pearl[9] that assigns weights to the hypothesis based on a square-error cost criterion. This influence is shown by the intensity of grayness in the Belief Network Window. Note that a target node has to be selected for these factors to be computed.

Display belief histogram determines whether the belief histogram for a node is displayed in the node/link display window when a node is selected.

Maximum number of values to be displayed allows the user to see the top *n* sorted values of the selected node (based on belief) in the node/link display pane. The user can click-left on this field to edit this value. A value of 0 in this field would display all of the values of a node. The default (0) is to display all of the belief values.

Load Pathname allows the user to provide a default pathname for the directory of files to be used. This should be given as a pathname such as:

/usr/prj/bart on a SUN

local:>bart>version3.0> on a Symbolics.

Delete-model :

Allows the user to delete the current model from the system. If there are any other models loaded into the system, then it presents a pop-up menu of these models allowing the user to choose a model to become the new current model.

Evidence :

Adds external evidence to or deletes the evidence from a node. After the user clicks on *Evidence* a pop-up menu appears with several possible choices: add, delete, read, or write, depending on whether evidence has previously been provided for this knowledge structure. The user should click-left on the appropriate choice. If the user wants to add evidence, he should click on *add* and then on the appropriate node. Now, another pop-up menu appears to add evidence using either a *Graphic scale* or *Numeric vector*. If the user chooses *Graphic scale* a menu appears which has the heading *New Likelihood Ratio* followed by the node's name. (A sample screen menu for input of external evidence is shown in Figure 2.) All of the values of the hypothesis are listed on the right hand side of

this menu. A scale is presented across the top with gradations of evidential support ranging from "Rule out" to "Affirms." The area under the scale heading has a double arrow for each value which is initially placed under the position indicating indifference (i.e. no evidential impact one way or the other). The user can indicate the level of support (from the new external evidence) for each value by placing the cursor in the appropriate position and then mouse-clicking left, causing the line with an arrow to be placed there. On the left of these scale areas are numbers representing the numeric value of the selection. The user has the option of choosing several modes for entering the evidence. The default scale has *Discrete* intervals, and the marker will be placed under the closest interval gradation to where the user clicks. The user can choose to make the scale discrete or continuous by clicking left on *Discrete* or *Continuous* to change the option. The numbers listed on top of the scale can be shown or hidden by clicking left on *Show-scale* or *Hide-scale* to change the option. The default is to show the scale. Mouse-click left on either *done* to enter the new evidence into the system or *abort* to ignore the change. The node to which external evidence has been added will be shown in reversed video (white name on a dark background) in the Belief Network Window. If the user chooses to enter the information as a *Numeric Vector*, a window will pop up to take a list of values. This window has two buttons: done and abort. After entering the information the user should click on one of these buttons. Clicking on done adds the given information as evidence to the selected node and clicking on abort aborts it. Once these values are entered, the effect of that evidence can be propagated by clicking on *Update*.

Read is used to read in a file of previous evidences for a particular knowledge structure. The filename prompt appears in the message window. The system then reads all the evidences present in this file and attaches them to the nodes in the network. Once the evidence is entered, the effect of that evidence can be propagated by clicking on *Update*.

If evidence was previously entered, *Delete* can be used to delete the evidence. The user should click on the node which he wants evidence deleted from. If there is only one piece of evidence for this node, that evidence will be deleted. If there are multiple pieces of evidence for this node, then a menu will pop up which lists each evidence and also the string "all." The user can click on the appropriate individual evidence he would like to delete or "all" to delete all evidence from this node. *Update* should then be selected to bring the network back into equilibrium.

Again, if evidence was previously entered, *Write* can be used to save the current evidences for the current knowledge structure in a file. A prompt for the file name appears in the message window.

Any of the above operations brings the network out of equilibrium and the network name appears in ordinary font in the global system parameters pane.

Exit-bart :

Exits from the program.

Load :

If more than one domain model is loaded into the system then a prompt would appear asking whether to load a new model or just to select one of the models already loaded. If the user chooses the option *existing* from this menu, then another menu of all the loaded models would appear so the user can select one of them. If the user wants to load a new model then it prompts for a data file and loads it. If the data file is in the current directory, then just the name of the file can be given at the end of the path provided. Otherwise, the pathname should be provided. A BaRT data file *fff* can be stored in 3 different files: fff-bart.lisp fff-bart-int.lisp and fff-bart-int.bin. When the user provides the .lisp extension, then the source file is loaded. When the user does not give an extension, the most recent of the internal binary, internal source and regular source is loaded in that order. When loading, BaRT performs all the necessary internal calculations, brings the networks into equilibrium, and displays the domain model. In case of a network with loops, BaRT first converts the given network into a singly connected network and internally creates this converted network as an auxiliary network. This auxiliary network is used in processing(updating) the given network with loops. If the compile data file option in the default modes is on and if the user is loading a source file, then besides loading the file, BaRT compiles the file in an internal format suitable for BaRT to load next time and compiles this file and saves it as *filename-bart-int.lisp/bin.*

Prior-prob :

Allows the user to provide prior probabilities for a selected top node in the current knowledge structure. If a node is selected then a temporary window appears to take the new prior probability and then changes the current prior probability of that top node to the newly given probability. This brings the network out of equilibrium.

Node-info :

After choosing this, clicking on any node (chance, decision, value, constraint, or query) displays the information about that node in the node/link display pane. Depending on which user modes have been selected, a histogram of the belief distribution for the values of a selected node may be displayed in addition to the actual belief and belief* vectors, the external evidence for the node, and a brief description of the node. A button *Link Info* would also appear in top right corner

of this pane. The user can then click on *Link Info* to see information displayed such as the λ, λ^*, π, and π^* vectors associated with the links of this node and the joint conditional probability matrices showing the relationship between the node and its parents. To see the additional conditional probabilities if there is more than one parent, click on *Next* in this window. Each entry in the matrix represents the probability that the manifestation (child) has the value indicated in that row given that the causes (parents) have the values indicated in that column. The link information is displayed in a pop up window on the left hand side of the screen (in the Belief Network Display pane.) When finished looking at the link information, click on *Link Info* again to get back the network display.

Query :

This allows the user to determine the current belief in a given boolean expression. After clicking on this, a window pops up in which the user can enter the name of the query node, a boolean expression such as:

(or (equal node-1 True) (equal node-2 False))

and the role. Note that the names of the nodes and the values are always case-sensitive. Please refer to Section 3.1 *(General Conventions)* for syntax. Then the user can *Update* to bring the network into equilibrium and then click on *Node-info* for the query node to see the current belief in this expression as a list of (true false) values. Note that this will not affect the belief distribution of the other nodes in the network.

refresh :

Refreshes the display.

Reset :

Resets the current knowledge structure back to the initial equilibrium state so the user can try a new run with new observations without loading and reinitializing. Any constraint nodes will be disabled at this point (note that they are no longer in boldface type), but they can be made active again by clicking on *Constraint* and then *Enable* in the pop-up menu.

Targetnode :

Allows the user to select the target node associated with the option *Compute measure of impact* within the *Default-modes* command. *Compute measure of impact* computes a factor at each node which is a measure of the uncertainty that can be reduced in the target node. This influence is shown by the intensity of grayness for each node in the Belief Network Window. The darker the gray, the more influence adding evidence to that node will have on the target node.

Update :

Brings the current knowledge structure into equilibrium and redisplays the information.

Zoom :

Allows the nodes in the current domain model to be increased or decreased in size. A menu pops up, and the user must click-left on the numeric field next to the label *zoom factor*. The user should always type a positive number to change the size of the nodes. These numbers always use the original size as the reference point, so the user should provide a number greater than one to increase the size (e.g., 2 will double the size), a number greater than 0 but less than one to decrease the size as compared to the original size, or *1* to return to the original size. The user must click-left on *done* to process the request or on *abort* to ignore the request. Note: when the scale factor is less than 1.0, the nodes drawn won't have any names.

5. Knowledge Acquisition

This version of BaRT provides a graphics user interface for entering information about the networks of a domain model. The user must enter the nodes of a network as a *unit*, as described for each of the types of networks below. For both belief networks and influence diagrams, each unit contains a current child node and all of its parents and the joint conditional probability tensor representing the relationship between the node and its parents. In the case of a Taxonomical hierarchy a unit is a node and its children. Note that the user does not need to specify links here. For taxonomical hierarchies there will not be any conditional probabilities between nodes. The belief of a node is the sum of the beliefs of its children. Each node here represents a subset of the set of mutually exclusive and exhaustive constituent elements over which the taxonomy is defined. Constituent elements (components) can be defined in the joint conditional probability (jcp) pane (which is used to take joint conditional probabilities for regular belief networks and influence diagrams).

The definition of a *unit* was chosen because it allows one to focus on a local contained relationship. For example, a node and all of its parents can be quantified locally by the joint conditional probability tensor in belief networks in the BaRT model. A complete network will then be a full distribution composed of these local unit relationships.

5.1. Compiling and Loading the BaRT KA module.

Symbolics: Get into a Common LISP environment with or without PCL. Edit the file *bart-defsys.lisp* to indicate the directory where BaRT files reside. From the LISP listener, load the file *bart-defsys*. If you just want to load the system, then type *(load-ka)*. In this case it loads the appropriate source or binary files of the system. Besides the BaRT files, the system may load a file *clos* depending on whether PCL is in the environment. Now, invoke the program by first pressing the *Select* key and then pressing the *Symbol, Shift* and *E* keys simultaneously.

Sun: Edit the file *bart-defsys.lisp* to indicate the directory where BaRT files reside. Invoke Common LISP with or without PCL in a sunview window, and then load the file *bart-defsys*. If you just want to load the system then type *(load-ka)*. In this case it loads the appropriate source or binary files of the system. Besides the BaRT files, the system may load a file *clos* depending on whether PCL is in the environment. Change to package *bart-frame-ka* by typing *(in-package 'bart-frame-ka)*. Now, invoke the window environment by typing *(start-window)*. This creates a lisp pane in an editor environment. Then invoke the program by typing *(start-bart-ka)* in that lisp pane.

To compile the system type *(load-ka t)*. This compiles and loads the system after compiling. It is strongly suggested that a disksave image be created for the Sun version. To do this after the files have been compiled and loaded, type the command *(disksave "Bart-KA" :restart-function #'bart-frame-ka::start-all :full-gc t)* before you invoke the window environment.

5.2. Knowledge Acquisition Windows and Commands

The screen consists of the following windows: the title pane with some command buttons and the network name, the current node window, the parent/children nodes window, the joint conditional probability window, the network display, and the messages window. The joint conditional probability window and the network display are scrollable in the standard way for the machine. Figure 3 provides a sample screen display. Note that all selections of buttons, commands, choices within a menu, or nodes should be made by *clicking left* on the appropriate field. Please refer to Section 3.1 *(General Conventions)*.

The *title window* contains the title *Knowledge Acquisition Tool*. Below the title on the right hand side is the name of the current network being edited, if there is one yet. Below the title on the left hand side are the following commands:

Default-modes

After clicking on this, a temporary menu entitled *Select Default Modes* of all the user settable options appears. The user can change any of these values by clicking left on them. The global options appear with their present values in boldface. This command can be terminated by clicking either on *done* to process the request or *abort* to ignore the request. The options available are

Write ka file while saving bart file makes the system write a file in ka format also when writing a file in bart format.

Data Pathname allows the user to provide a default directory for file pathnames. This should be given as the directory-name portion of a pathname such as:

/usr/prj/bart/ on a SUN.

local:>bart>version3-0> on a Symbolics.

Delete-network

Allows the user to delete the current network if any. If more than one network is loaded, then it lets the user choose a network and makes that network current. Note that this will REMOVE the current network permanently from the system.

edit-network :

Allows the user to edit a new or existing network (which was defined or loaded during this session.) If no network exists yet, a menu appears with choices for the type of network: *Belief Network, Influence Diagram* or *Taxonomic Hierarchy*. If there is already at least one network, then the choices for a *new* network and each of the existing networks appear. If the user clicks on *new*, then the above three choices for the type of network appear. When entering a new network, the user must enter a name for this network in the messages pane. Simply type the name followed by a Return. Now the name appears in the upper right hand corner of the screen. The current network would be the chosen or new network.

load-file :

Allows the user to load an existing file which has been saved for future editing in Knowledge Acquisition mode. A prompt for "Filename:" followed by the pathname of the default directory appears in the messages screen. The user should supply a new pathname, if necessary, with the name of the file. If the networks within this file exist already in the current editing session, the user must choose whether to replace each of the current networks with the networks in the file or ignore the load. Note: a newly loaded network may not be the current network unless the current network is being replaced or there is no current network.

save-file :

Prompts the user whether to save the networks in bart format or ka format or both. Then it presents all the network names and a model name in case of bart/both format and then saves those networks selected by the user in the chosen

format. The chosen networks in bart format are saved in a file *fff-bart.lisp* and in ka format are saved in a file *fff-ka.lisp* where *fff* is the name supplied by the user. Note: if the networks are not fully defined, then it can't save them in bart format. In that case it gives an error message indicating that the network is not fully defined.

exit :

Allows the user to exit from the knowledge acquisition tool.

The *Current Node pane* contains buttons necessary to edit the current unit as a whole. It displays the current node. The following commands are available in this window:

abort :

The current *edit-unit* session will be aborted, and the network will be restored to its previous state.

delete :

Allows the user to delete the current unit.

done :

The information of the current unit will be stored and the network will be redrawn.

edit-unit :

Allows the user to create a new unit or modify an existing unit. After the user clicks on this command, a pop-up menu will appear if there are existing nodes in the network display pane. The user should click left on either *new node* or *existing node* depending on whether he wants to create a new unit or modify an existing unit. If existing node is chosen, then the user should mouse click on the node that he wants to modify. Note that this pop-up menu will not appear if there are no existing nodes yet. Depending on the type of network and the type of the node several fields and a button would appear in this pane. All these fields can be modified by mouse clicking on the value that appears in front of the field. Clicking on the buttons would execute some function associated with that button to produce some side effect. These fields and buttons are explained below.

Name: This field contains the name of the current unit. If this is a new node, then this field initially has a unique default name.

Role: This is the documentation attached to this current node. The default value for this field is a null string.

Values: This field represents the values that the current node can take. This won't appear if the node is a value node or if the current network is a taxonomic hierarchy. The default value of this field is boolean (True False).

Topnode: This is a button which toggles the current node to a top node or an ordinary node. This is applicable to only belief networks and influence diagrams except for decision nodes. Clicking on this button provides/removes another field *Prior*. If the current unit has parents, then making the current unit a top node generates an error. The user must delete all the parents before making the current node a top node.

Prior: This represents the prior probability of the current node and consists of a list of positive numbers between 0 and 1.0 that sum to 1.0. The number of elements in this list should be equal to the number of elements in the value field.

Tree Name: This represents the name of the tree the current node belongs to and is applicable only for taxonomical hierarchies. Every hierarchy is defined as a collection of one or more taxonomies.

Leaf node: This button is a toggle which makes the current node in a taxonomic hierarchy an ordinary node or a leaf node. Making the current node a leaf node would display the *Components* field in this pane. Again, as in topnode, the user can make the current node a leafnode only if it doesn't have any children. Otherwise it generates an error. The user must delete all the current node's children before making it a leaf node.

Components: This is the subset of components that the current node is consisting of. If no components are supplied then the system assumes this leaf node is a component with a weight equal to 1.0. See also *Edit component*.

node-type :

Allows the user to switch among node types when editing an *Influence Diagram*. A menu will appear with the choices *chance node, value node* or *decision node*. Note that the appropriate input fields for information concerning this node will appear after the choice is made. Also note that no more than one value node is allowed in an influence diagram.

test-unit :

Allows the user to test the belief updating of a unit given the prior probabilities for the top nodes of this unit, the conditional probabilities of the current node with respect to its parents, and the evidence vector (i.e., the vector of current

evidence in support of each of the values of the effect node). After the user clicks on this command, a pop-up window will appear in the bottom right hand side of the screen. There will be slots for changing the prior for each top node and the Evidence Vector. The conditional probabilities appear in the joint conditional probability (jcp) pane and can be changed by the user. These conditional probabilities are used along with the user editable prior/evidence fields in the pop up menu to find the belief distribution of the current node. This is done whenever the user clicks on *Try* in the menu. The belief distribution is displayed as a vector and as a histogram. The standard deviation of the belief values is also displayed. The prior, evidence vector and the conditional probability entries can be changed for further testing or the user can click on *Done* to stop testing. Note that this is valid only for nodes in belief networks and influence diagrams except decision nodes.

The *Parent/Child* Nodes pane contains buttons necessary to edit the parents/children of the current unit. This pane presents information about the current nodes children/parents and also provides three menu commands applicable for this pane. The information displayed and the menu commands provided here depend on the type of the current network. In case of taxonomic hierarchy the current nodes children would appear in this pane and the menu contains add-child, delete-child and next child. Otherwise the menu contains add-parent, delete-parent and next-parent and the current nodes parents would appear in this pane. The information displayed in this pane is described below.

Name: This is the name of the current parent or child.

Values: This field contains the values that this current parent takes and is not valid for taxonomic hierarchies.

Tree Name: This field represents the tree name to which this current child belongs to and is applicable only for taxonomic hierarchies.

Note: All the fields are editable if the current parent/child is being created now and are not editable if it is a previously created node.

The commands that may appear in this pane's menu are:

add-parent :

Allows the user to add a new or an existing parent. A menu appears with the choices of new and existing (unless there are no existing nodes in the Network Display yet.) If the user clicks on *existing node*, he should then click on the

appropriate node which would then appear in this pane. Otherwise a new node with default values will automatically appear in this pane. Note that the user will not be able to change the information fields for an existing node chosen as a parent. If the user would like to do this, this node should be chosen as the *Current Node* in a new *edit-unit* session. This would appear only if the current network is an influence diagram or a belief network.

delete-parent :

Allows the user to delete the parent currently displayed in the *Current Nodes* window from the current unit. Note that if this node is also part of other units, it will remain attached to those other units. This would appear only if the current network is an influence diagram or a belief network.

next-parent :

Cycles through and displays the parents of the current unit. This would appear only if the current network is an influence diagram or a belief network.

add-child :

Allows the user to add a new or an existing child. A menu appears with the two choices of new and existing (unless there are no existing nodes in the Network Display yet.) If the user clicks on *existing node*, he should then click on the appropriate node which would then appear in this pane. Otherwise a new node with default values would appear in this pane. Note that the user will not be able to change the information fields for an existing node chosen as a child. If the user would like to do this, this node should be chosen as the *Current Node* in a new *edit-unit* session. This would appear only if the current network is a taxonomic hierarchy.

delete-child :

Allows the user to delete the child currently displayed in this pane from the current unit. Note that if this node is also part of other units, it will remain attached to those other units. This would appear only if the current network is a taxonomic hierarchy.

next-child :

Cycles through and displays the children of the current unit. This would appear only if the current network is a taxonomic hierarchy.

The *joint conditional probability (jcp) pane* at the bottom left of the screen is used differently depending on the kind of current network. It has a display pane and a menu

pane and the commands that appear in this menu are different and are explained here. If the current network is a taxonomic hierarchy then the menu pane consists of 4 buttons: *Delete component, edit component, Ignore,* and *Save.*

Delete Component :

This lets the user select a component from the display using the mouse and deletes that component.

Edit component :

This button allows the user to either create a new component or edit an existing component. If there are any components previously defined, then a pop-up menu appears letting the user select whether he wants to edit an existing component or a new one. In the case of an existing component, the user mouse selects the component that needs to be edited. Otherwise a new component appears with some defaults. The display pane originally before editing any component, displays all the components defined so far. Once the user creates/edits a component, it displays two fields: *Name* and *Weight* in this pane. The values of both the fields can be changed by mouse clicking on them.

Name: This field contains the name of the current component.

Weight: This field contains a number representing the weight of that component. The initial weight is always 1.0. Once the network is fully defined, all the weights associated with the components are normalized and used as prior probabilities.

Ignore :

Ignores the current component and any changes made would be ignored.

Save :

This saves the current component.

If the network is not a taxonomic hierarchy then the following is applicable. The Conditional Probabilities pane contains buttons necessary to edit the conditional probability tensor of the current unit. Once the user selects a way (explained later in the commands) to give information about the conditional probabilities, a table of mouse sensitive fields, with proper headings appear in this pane. Each field can take a number between 0 and 1.0. The values of the child (manifestation) node are always listed on the left as row labels for a matrix. The values of the parent nodes are on the top as column labels for the matrix. An entry in the matrix indicates the probability that the child node has the value listed in that row given that the parent nodes have the values listed in that column. When clicking left on any of the matrix entries to change them, the user enters the editing phase. Then he can enter a new number. Note that the user

doesn't have to kill the previously displayed value to enter a new number as is the case everywhere else while editing a field. The sum of all the numbers in each column (ie. the total probability of a particular child state given a set of parent states) must always be 1.0 (or within the interval [1-ε, 1+ε]). Initially all the fields of a column have default values. Default values are enclosed by [] brackets. Once the user fills one field, then the remaining fields in that column are always adjusted so that the total would be 1.0. This is done by distributing the residue (1.0 - sum of all the user given numbers in that row) equally to default values. If the sum of all the user given numbers in a column is more than 1.0, then all the numbers in the column would be reset to the default value of 1.0 divided by the number of entries in that column. To get to the next entry in this matrix or just to finish the input for the present entry, hit *Return*. To get to another specific entry *click left* on that entry. The user *must click middle or right* to come out of the edit phase. Since all of the table may not be visible in this window, the window scrolls automatically to keep the current field and its neighbors visible.

This pane contains six commands: *Joint-prob, Pairwise, Values, Noisy-gate, Set-vals, Next*. The user can provide the information about conditional probabilities using these commands. These are explained below:

Joint-Prob :

Allows the user to enter the complete joint conditional probability tensor between the current node and all of its parent nodes. A reasonable parent (preferably one with 3 or 4 values) is picked as the focus, meaning that the values for the other parents are set while the values for the focus parent vary in each sub-matrix displayed. To get to the next sub-matrix where the other parents have the next set configuration of values, the user should click left on the *Next* button. Please see the function of that button. This command is not valid for a value node in an influence diagram.

Pairwise :

Allows the user to enter conditional probabilities between the current node and each of its parents. Then these individual conditional probabilities are combined at save time using the type of gate (see *Noisy-gates*) selected by the user to form the joint conditional probability distribution. This command is not valid for a value node in an influence diagram.

Values :

Allows the user to give value information about the value node in an Influence Diagram for each state of its parents. Valid only for value nodes in an influence diagram. In this case the numbers in the jcp pane need not be between 0 and 1.0.

Selecting any of the above three commands would bring a matrix of values in to the joint conditional probability pane. Depending on the number of parents, only part of the full matrix (distribution) is brought into this pane. The next three commands are useful in filling some values in this matrix or to choose a gate or to go to the next sub-matrix and are explained below.

Noisy-gate :

When the user chooses to give pairwise conditional probabilities, a type of gate is required to combine them to form the joint conditional probabilities. This button allows the user to choose the gate type from the set {*Or, And, And+, And-*}. The *Or* gate is to specify that any one parent state can cause a change in the value of the current node irrespective of the states of the other parents. The *And* gate assumes that the parents are totally independent and there is no interaction between them. These two gates are explained in detail in [9]. The other two gates *And+, And-* are used to indicate highly positive and highly negative correlation between parent states. These gates are explained in [12]. The default gate is *And*.

Set-vals :

Once a submatrix is present in the jcp pane, the user can use this command to change some of the fields. A pop-up menu containing *Identity, uniform,* and *expression* would appear. The user can select one of these to change the fields.

Identity:

This would appear in the menu only if the number of rows and the number of columns of the submatrix present in the jcp pane are equal. Selecting this would change the submatrix to an identity matrix.

Uniform:

Selecting this would change the values of all the entries in the present submatrix to the value of 1.0 divided by the total number of entries in that column.

Expression:

This would present a variable value menu with 15 slots to take equations and an exclusive choice of *Local* and *Global*. The Local, Global slot is to indicate whether the change should take place only in the submatrix present in the window or to the whole underlying matrix. This is applicable only if the user is giving joint conditional probabilities. For pairwise probabilities, this won't be present. Each of the equations field takes an equation and a number between 0 and 1.0. For example, "(& (= Coma True) (~ (= Calcium False))) .5" means that if Coma is True and Calcium is not False then set the value of that entry to 0.5. The system goes through these equations in the order given and sets the value of each entry to the number associated with the first satisfied equation. If none of the equations are satisfied, then it won't change that entry. Optionally just a number can also be given as the last equation, in which case that number would be used if none of the previous equations are satisfied for an entry as a true clause in a

cond statement. Once the entries are changed, then each column is normalized and these are the values that would be present in the jcp pane.

Instead of typing a number(same) in each entry, the above 3 methods allow the user to change several values in the jcp pane. Then if the user wants to change a particular entry, he can click left on that entry to change it.

Next :

This button is used to cycle through the next submatrix of values to change, and to change the focus and configuration. Clicking left would put the next submatrix to be changed in the window. Clicking middle allows the user to change the focus/configuration. In case of pairwise probabilities, a pop-up menu of the remaining parents would appear and the user can select one of them. The selected parent would be the new focus. In case of joint probabilities, another menu appears where the focus or the configuration of the other parents can be changed. Configuration can be changed by clicking on the boxes with arrows in them. The focus can be changed by clicking on the *Change-focus* button. Then a pop up menu appears with all the parent names and the user can select one of them as the new focus. Finally click on *Done* to get back to editing the probabilities.

The *network display pane* is on the bottom right hand side of the screen occupying a large portion of the screen. This pane is used to display the current pieces of the current networks (subnetworks) which consist of nodes and their links. The following commands allow the user to move the nodes' positions:

Redraw :

Refreshes the screen.

Move-nodes :

Allows the user to move one or more nodes to a different location. After clicking on the command, the user should click-left on the appropriate node(s) and then click-middle on the spot where the first node selected by the mouse left click should be placed. Clicking right aborts this command.

Move-subnet :

Allows the user to move one subnet to a different location. After clicking on the command, the user should click-left on a node of the subnet and then click-middle on the spot where the node (and all its children in that subnet) should be placed. Clicking right aborts this command.

In the bottom of the screen is the *Interaction window* for normal interaction. Lisp expressions can be evaluated in this pane on the SUN. On the Symbolics, the user should press the *SUSPEND* key to get a top level lisp read-eval loop in this pane to try any lisp expressions. During this time, none of the commands are active. The user should press the *CONTINUE* key to get back from the suspend mode.

The *message pane* at the bottom of the screen displays messages when necessary (i.e., if something is incorrect or if something might be unclear). Prompts for file names will also appear here.

6. Example

This example will show the user how to build a medical information network using the knowledge acquisition module and then how to load this into BaRT and run the system by using various commands. Refer to the previous sections on BaRT commands and Knowledge Acquisition commands while doing this. See Figure 3 for a picture of the network being defined.

First, invoke the knowledge acquisition module (see chapter 5). Click on *edit-network* and then on the choice *Belief Network* in the pop up menu. Now enter the Network Name *med1* in the messages buffer. At this point, the user can start entering information at any point in the network that he is comfortable with. We will start with the top nodes and work our way down. Remember a unit is a node and all of its parents, so a top node is a one-node unit. Click on *edit-unit* to provide the only top node in the Effect Node pane. Now the Effect Node pane contains 3 fields with their names and default values and a button. Position the mouse on the field in front of the Name. A box will highlight this field. Click on this field. Press *<cntr> k* to remove the existing default value and enter the name *Cancer* and hit return. There is no need to provide values here; the default of *True False* is what we would like here. Click on the role field and enter a brief description of the node if you would like. Now click on the *top node* button which presents the *Prior Probs* field. Click on this field and enter the list of prior probabilities for this node *(0.2 0.8)*. Now click on *Done*. The network pane will now display this node.

Now we will enter the node *Tumor*. Again, click on *edit-unit* and then on *New Node* in the pop up menu. Click on the name field and enter the name *Tumor*. Again, the default for the Values field is appropriate. Now click on *Add-parent* in the Cause Nodes pane. Click on the choice *existing node* since the parent has already been defined and then click on the *Cancer* node in the network pane. Now click on the choice *Joint-Prob* in the Conditional Probabilities window. Now the JCP window contains a table of mouse sensitive default joint conditional probabilities. Enter *0.2 0.8* in the first column and *0.05 0.95* in the second column. To enter these numbers, just click on the appropriate entry in the matrix. The chosen cell would be displayed in reverse video indicating that it is in edit mode so the user can enter a new value in that cell. Enter a new number and hit Return. Then the system would accept the new number (if it is a valid number) and updates the other default values in the same column so that the total would add up to 1.0 (or within the interval $[1-\varepsilon, 1+\varepsilon]$). Then the system positions itself on the next cell for the user to enter a new number. The user can hit return without giving any new number to keep the existing value. In that case, the system moves to the next editable cell for the user to modify. The numbers entered mean that the probability that the patient has a tumor given that the patient has cancer is .2, the probability the patient does not have a tumor given that the patient has cancer is .8, etc. After the new values are entered, click on middle or right to get out

of the current edit mode. This pane is a scrollable pane and can be moved with the mouse to see other values. Once the conditional probabilities are entered, click on *Done* in the Effect Node menu pane. This would create another node *Tumor* connected to the previously created *Cancer* node in the network pane.

Similarly, enter the node *Calcium*. Leave the values as True False, and add the existing parent *Cancer*. Here the Joint Conditional Probabilities are *0.8 0.2* for the first column and *0.2 0.8* for the second column.

Now we will enter the node *Coma*. Enter the name, values (which can be left at the default) and role (optional). Now, add the two existing parents *Calcium* and *Tumor*. Now click on *Joint-Prob*. The values of *Calcium* will vary as one value of *Tumor* is held constant for each piece of the matrix to be entered. First, the value of *Tumor* is set at True. Enter *0.8 0.2* in the first column. and *0.8 0.2* in the second column.

Instead of entering these numbers straight, let's use some of the system's capabilities to fill them. Click on *Set-vals* and click on *identity* in the pop-up menu. Observe that the matrix displayed is an identity matrix in the jcp pane. Again click on *Set-vals* and click on *Uniform* in the pop-up menu. See that all the values in matrix are equal. Click on *Set-vals* and click on *Expression* in the pop-up menu this time. This would present a pop-up menu with 15 expression fields, and an exclusive choice of local or global. The system creates the conditional probability matrix satisfying the given expressions. When more than one parent is present, the jcp pane presents only a sub-matrix instead of the entire matrix of conditional probabilities. The local and global option is useful to specify whether to fill the sub-matrix displayed or the entire underlying matrix of conditional probabilities satisfying the given the expressions. The expressions internally form a cond statement. Each expression field takes an expression and a number. Optionally one can have just a number without an expression as the last expression indicating the default number in case all the expressions given before fail. Initially the jcp presented in the menu has some (previous or default) values. The system evaluates each expression in the given order (from top to bottom) and changes those entries satisfying the first expression from the top and fills those cells with the value associated with that expression. The syntax of this expression is explained in Section 3.1 *(General Conventions)*. In the present case type these two expressions in this pop up menu. Choose locally to change the entries only for the sub-matrix displayed in the jcp pane. The two expressions are

(= Calcium True) .8
(= Calcium False) .2

After giving the expressions, and choosing the local option, click on *Done*. This should update the sub-matrix in the jcp pane to have (.8 .2) in each column. Then click on *Next*. Now, the value of *Tumor* is set at False. Enter *0.8 0.2* in the first column, and *0.05 0.95* in the second column. Then click middle or right. Then click on *Done* in the Effect node pane menu.

Now enter the last node *Headaches*. Again enter the name, values (leave as True False), and role. Add the parent *Tumor*. Now click on *Joint-Prob* and add the Joint Conditional Probabilities by entering *0.8 0.2* in the first column, and *0.6 0.4* in the second column. Then click middle or right and then click on *Done*.

Now that the network is complete, save it for running in BaRT. Click on *save-file* and select *Both* in the pop up menu and then type in the name *medical.lisp* (followed by a Return) as the file name in the messages window. The system would present another menu with all the networks defined so far and with a *Model Name*. Fill in the model name with *Cancer-diag* and select the network (only one defined so far). Then click on *Done*. Then the system would store two files: *medical-ka.lisp* suitable to load into the KA for future editing, and *medical-bart.lisp* suitable to load into BaRT. Note that throughout this editing, the system complains whenever it sees an error and gives the user an opportunity to correct it. Now exit the knowledge acquisition module by clicking on *exit*.

Now, invoke the BaRT module (see chapter 4). Now click on *load* and enter *medical* for the file name prompt at the bottom of the window. The system then loads the medical-bart.lisp file. Initially BaRT performs a number of internal calculations and brings the network into equilibrium. If any of the networks in the file contain loops, it also converts them into singly connected networks. After this initial processing is done it saves the states of the networks loaded into a file medical-bart-int.lisp first and then compiles it and saves the binary file in *medical-bart-int.lbin*. This way the system loads this internal file next time without the initial processing overhead. BaRT saves these internal files depending on the system option *compiler data file*. After loading the file, the system presents the network in the network display pane.

First, look at the top node in the network, *Cancer*. (Click on *Node-info* and then on the node *Cancer*.) Now we will provide some evidence to a node in the network. Click on *Evidence* and then on *Add* from the pop up menu provided. Now click on the node *Headaches* in the Belief Network Display window. Click on the choice *Likelihood Ratio*. Now a window pops up with a scale for entering support for the various values of this node. Say that the patient has indicated that he does indeed have frequent headaches. So, click all the way on the right of the scale for Affirms in the row for True and click all the way on the left of the scale for Ruleout in the row for False. Click on *Done* in this window. Note that the node *Headaches* is now highlighted to show it has evidence attached. Also, the Current Knowledge Structure is no longer in boldface type, showing the network is not in equilibrium. Now click on *Update* to propagate the effects of the new evidence. Now look at the *Cancer* node again in the *Node/Link information display pane* and note that the belief in it has changed.

Now let's add a constraint node to this network. Click on *Constraint* and then *Add* from the pop up menu provided. A window appears for providing information for this new node. Click on the name slot (to the right of the word *Name*) and type in the name *Cnstr1*. Click on the Expression field and type in the expression: (or (equal Coma False) (equal Calcium True)) to indicate that either the patient is not in a coma or he has a high calcium level. This is a boolean expression set to True which now constrains the network. Now Click on *Done*. Click on *Update* to propagate the effects of this constraint through the network. Note the effect on the beliefs in the values of the *Cancer* node. Now disable this node by clicking on *Constraint*, then *Disable* in the pop up menu, and then the Constraint node. Note that this node is no longer in boldface type. Click on *Update* and note that the beliefs in the values of the *Cancer* node revert to the previous values. Try enabling the constraint again, and then click on *Node-info* and then on the constraint node to see that this node is indeed an active node once again. Now click on *Reset* and note that the evidence is removed from this network and the constraint node is disabled. Try adding more evidence, constraint nodes and query nodes and play with removing some of them, noting the effects on beliefs in the values of various nodes.

The current version of BaRT comes with an example data file called example1.lisp (example1-ka.lisp and example1-bart.lisp) which also includes this medical network we defined here. It is a domain model with three belief networks. The user is strongly recommended to load this file and try out the different commands on it.

References

1. Booker, L. B. and Hota, N., Probabilistic Reasoning About Ship Images. In J. Lemmer and L. Kanal (Eds.) *Uncertainty in Artificial Intelligence 2*. Amsterdam: North-Holland, 1988.
2. Booker, L. B., Hota, N. and Hemphill, G., Implementing a Bayesian Scheme for Revising Belief Commitments. Proceedings of the 3rd AAAI Workshop on Uncertainty in Artificial Intelligence, Seattle, WA, July 10-12, 1987, p.348-354.
3 Cooper, G. F., A Method for Using Belief Networks as Influence Diagrams. *Proceedings of the Fourth Workshop on Uncertainty in Artificial Intelligence* (pp. 55-63), Minneapolis, MN, 1988.
4. Kim, J., CONVINCE: A CONVersational INference Consolidation Engine. Ph.D. Dissertation, University of California, Los Angeles, 1983.
5. Kim, J. and Pearl, J., A Computation Model for Combined Causal and Diagnostic Reasoning in Inference Systems, *Proceedings of IJCAI-83*, Los Angeles, CA, August 1983, p.190-193.

6. Pearl, J., Fusion, Propagation, and Structuring in Belief Networks, *Artificial Intelligence*, Vol 9, p.241-288, 1986.

7. Pearl, J., Distributed Revision of Belief Commitment in Multi-Hypotheses Interpretation. Proceedings of the 2nd AAAI Workshop on Uncertainty in Artificial Intelligence, Philadelphia, PA, August 8-10, 1986, p.201-209.

8. Shafer, G., Tversky, A., Languages and Designs for Probability Judgement. Cognitive Science, Vol 9, p.309-339, 1985.

9 Pearl, J., *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. San Mateo, CA: Morgan Kaufmann, 1988.

10 Chang, K. and Fung, R., Node Aggregation for Distributed Inference in Bayesian Networks. *Proceedings of the Eleventh International Conference on Artificial Intelligence*. Detroit, MI: Morgan Kaufmann, 1989.

11. Geffner, H. and Pearl, J., Distributed Diagnosis of Systems with Multiple Faults. Technical Report CSD-860023, Computer Science Department, University of California, Los Angeles, CA, December 1986.

12 Bonissone, P. and Decker, K., Selecting uncertainty calculi and granularity: an experiment in trading off precision and complexity. In L. Kanal and J. Lemmer (Eds.) *Uncertainty in Artificial Intelligence*. Amsterdam: North-Holland, 1986.

Appendix A

Tensor Product Computation

A *tensor* is a mathematical object that is a generalization of a vector to higher orders. The order of a tensor is the number of indices needed to specify an element. A vector is therefore a tensor of order one and a matrix is a tensor of order two. Three standard operations defined on tensors are relevant to this discussion:

Term Product The term product is defined between two tensors **A** and **B** having the same indices. Each element in the resulting tensor **C** is simply the product of the elements with the corresponding indices from **A** and **B**.

$$\mathbf{C} = \mathbf{A} * \mathbf{B} \quad \text{where} \quad c_{i_1 \cdots i_n} = a_{i_1 \cdots i_n} \times b_{i_1 \cdots i_n}$$

Outer Product The outer product of two tensors **A** and **B** having order m and n respectively is a tensor **C** of order $m+n$. Each element of **C** is the product of the elements of **A** and **B** whose aggregate indices correspond to its own indices.

$$\mathbf{C} = \mathbf{A} \circ \mathbf{B} \quad \text{where} \quad c_{i_1 \cdots i_m j_1 \cdots j_n} = a_{i_1 \cdots i_m} \times b_{j_1 \cdots j_n}$$

Inner Product The inner product of two tensors **A** and **B** is a tensor formed by taking the outer product of **A** and **B** and then summing up over common indices that appear both in **A** and **B**. If **A** is of order m, **B** is of order n and they have k common indices then the inner product **C** is a tensor of order $(m-k)+(n-k)$.

$$\mathbf{C} = \mathbf{A} \bullet \mathbf{B} \quad \text{where} \quad c_{i_1 \cdots i_{m-k} j_1 \cdots j_{n-k}} = \sum_{l_1, \cdots l_k} a_{i_1 \cdots i_{m-k} l_1 \cdots l_k} \times b_{l_1 \cdots l_k j_1 \cdots j_{n-k}}$$

Equations

Let $\boldsymbol{\lambda}_{Y_j}$, $\boldsymbol{\lambda}^*_{Y_j}$, $\boldsymbol{\pi}_{U_i}$, and $\boldsymbol{\pi}^*_{U_i}$ be vectors (or, equivalently, tensors of order 1) whose elements are the messages a node X receives from its children and its parents respectively:

$$\boldsymbol{\lambda}_{Y_j} = \left[\lambda_{Y_j}(x_1), \cdots, \lambda_{Y_j}(x_r)\right] \quad \text{where } r \text{ is the number of possible values for } X$$

$$\boldsymbol{\lambda}^*_{Y_j} = \left[\lambda^*_{Y_j}(x_1), \cdots, \lambda^*_{Y_j}(x_r)\right] \quad \text{where } r \text{ is the number of possible values for } X$$

$$\boldsymbol{\pi}_{U_i} = \left[\pi_X(u_{i_1}), \cdots, \pi_X(u_{i_{r(i)}})\right] \quad \text{where } r(i) \text{ is the number of possible values for } U_i$$

$$\boldsymbol{\pi}^*_{U_i} = \left[\pi^*_X(u_{i_1}), \cdots, \pi^*_X(u_{i_{r(i)}})\right] \quad \text{where } r(i) \text{ is the number of possible values for } U_i$$

The term product of all $\boldsymbol{\lambda}_{Y_j}$ vectors is another vector $\boldsymbol{\Lambda}$ of length r given by

$$\boldsymbol{\Lambda} = \boldsymbol{\lambda}_{Y_1} * \cdots * \boldsymbol{\lambda}_{Y_m} = \left[\prod_{j=1}^{m} \lambda_{Y_j}(x_1), \cdots, \prod_{j=1}^{m} \lambda_{Y_j}(x_r)\right]$$

The term product of all $\boldsymbol{\lambda}^*_{Y_j}$ vectors is another vector $\boldsymbol{\Lambda}^*$ of length r given by

$$\boldsymbol{\Lambda}^* = \boldsymbol{\lambda}^*_{Y_1} * \cdots * \boldsymbol{\lambda}^*_{Y_m} = \left[\prod_{j=1}^{m} \lambda^*_{Y_j}(x_1), \cdots, \prod_{j=1}^{m} \lambda^*_{Y_j}(x_r)\right]$$

The outer product of all $\boldsymbol{\pi}_{U_i}$ vectors is a tensor Π of order n given by

$$\Pi = \boldsymbol{\pi}_{U_1} \circ \cdots \circ \boldsymbol{\pi}_{U_n} \text{ where } \pi_{k_1 \cdots k_n} = \prod_{i=1}^{n} \pi_X(u_{i_{k_i}})$$

The outer product of all $\boldsymbol{\pi}^*_{U_i}$ vectors is a tensor Π^* of order n given by

$$\Pi^* = \boldsymbol{\pi}^*_{U_1} \circ \cdots \circ \boldsymbol{\pi}^*_{U_n} \text{ where } \pi^*_{k_1 \cdots k_n} = \prod_{i=1}^{n} \pi^*_X(u_{i_{k_i}})$$

We can consider the set of fixed probabilities $P(x \mid u_1, \cdots, u_n)$ as elements of a tensor $\mathbf{P}$ of order $n+1$. Now if we compute the inner product of $\mathbf{P}$ with Π we obtain a tensor of order 1 (the indices for the U_i are common to both tensors):

$$\mathbf{P} \bullet \Pi = \left[\sum_{i_1, \cdots, i_n} P(x_1 \mid u_{i_1}, \cdots, u_{i_n}) \prod_{k=1}^{n} \pi_X(u_{i_k}) \, , \cdots , \sum_{i_1, \cdots, i_n} P(x_r \mid u_{i_1}, \cdots, u_{i_n}) \prod_{k=1}^{n} \pi_X(u_{i_k}) \right]$$

If we make the summation operator explicit, we can rewrite the formula as

$$\mathbf{P} \bullet_+ \Pi = \left[\sum_{i_1, \cdots, i_n} P(x_1 \mid u_{i_1}, \cdots, u_{i_n}) \prod_{k=1}^{n} \pi_X(u_{i_k}) \, , \cdots , \sum_{i_1, \cdots, i_n} P(x_r \mid u_{i_1}, \cdots, u_{i_n}) \prod_{k=1}^{n} \pi_X(u_{i_k}) \right]$$

We can now denote the formula for BEL as

$$\mathbf{BEL} = \alpha \, \Lambda * (\mathbf{P} \bullet_+ \Pi)$$

If we compute the inner product of $\mathbf{P}$ with Π^* we obtain a tensor of order 1:

$$\mathbf{P} \bullet \Pi^* = \left[\sum_{i_1, \cdots, i_n} P(x_1 \mid u_{i_1}, \cdots, u_{i_n}) \prod_{k=1}^{n} \pi^*_X(u_{i_k}) \, , \cdots , \sum_{i_1, \cdots, i_n} P(x_r \mid u_{i_1}, \cdots, u_{i_n}) \prod_{k=1}^{n} \pi^*_X(u_{i_k}) \right]$$

The BEL^* computation requires us to maximize over all elements u_k rather than taking a sum, so we can redefine the inner product operator as a maximize operator, and we can denote this new inner product with the symbol $\bullet_{\max}$.

$$\mathbf{P} \bullet_{\max} \Pi^* = \left[\max_{i_1, \cdots, i_n} P(x_1 \mid u_{i_1}, \cdots, u_{i_n}) \prod_{k=1}^{n} \pi^*_X(u_{i_k}) \, , \cdots , \max_{i_1, \cdots, i_n} P(x_r \mid u_{i_1}, \cdots, u_{i_n}) \prod_{k=1}^{n} \pi^*_X(u_{i_k}) \right]$$

Now the $BEL^*(x)$ computation can be written in tensor notation as

$$\mathbf{BEL}^* = \alpha \, \Lambda^* * (\mathbf{P} \bullet_{\max} \Pi^*)$$

Moreover, it is clear that we can use similar methods to compute the messages that node X will send to its neighbors. The vector $\boldsymbol{\pi}_{Y_j}$ destined for child Y_j can be computed by term-by-term division of the elements of **BEL** by the elements of $\boldsymbol{\lambda}_{Y_j}$, and the vector $\boldsymbol{\pi}^*_{Y_j}$ can be computed by term-by-term division of the elements of $\mathbf{BEL}^*$ by the elements of $\boldsymbol{\lambda}^*_{Y_j}$. The vector $\boldsymbol{\lambda}_X$ destined for parent U_i can be computed just like **BEL** except that we replace the vector $\boldsymbol{\pi}_{U_i}$ with a unit vector $(1, \cdots, 1)$ of equal length when computing the outer product Π, and the vector $\boldsymbol{\lambda}^*_X$ can be computed just like $\mathbf{BEL}^*$ except that we replace the vector $\boldsymbol{\pi}^*_{U_i}$ with a unit vector $(1, \cdots, 1)$ of equal length when computing the outer product Π^*.

Appendix B - Using the System without the Graphic Interface

BaRT can be run without the graphic interface. The user must get into a Common LISP environment with or without PCL. Load the file *bart-defsys.lisp*. Then type *(load-bart-gen t)* to load the system. This loads all the required files. The optional argument is to specify whether to compile files where needed. Along with the bart system this may load a file *clos* depending on whether PCL is already loaded. The bart system files that would be loaded are *pkgdefs, bart-util, bart-loops, bart, bart-rw, bart-frame-tty*. Now, type the command *(in-package 'bart-frame)*.

After loading a data file using *gen-load*, the following LISP functions can be invoked. All these functions take optional arguments and in the absence of these arguments, the system would prompt the user. The description of these functions are the same as the descriptions of the corresponding bart commands presented in chapter 4. Instead of using a graphic interface, these commands just prompt and take a value where needed. All the functions have a "gen" prepended to the corresponding command name. If the bart window interface command has subcommands then the corresponding functions are given here. For example, the command *constraint* has 4 subcommands: *Add, Delete, Enable, Disable*. The corresponding 4 functions are *gen-add-constraint, gen-delete-constraint, gen-enable-constraint, gen-disable-constraint*. All net/node/link/model/value names, when passed as arguments, must be strings although the user can omit the quotes when the system prompts for these values. A brief description of these functions is given here.

gen-active-structure :

Optional Arguments: a network name(knowledge structure name) which is of type string.
Synopsis: (gen-active-structure)
Description: This function allows the user to make a given knowledge structure current in the current model.

gen-add-constraint :

Optional Arguments: a constraint name and an expression. Both are of type string.
Synopsis: (gen-add-constraint "cnst1" "(& (= nd-A val2) (= nd-B val5))")
Description: This function adds a constraint satisfying the given expression.

gen-delete-constraint :

gen-enable-constraint :

gen-disable-constraint :

Optional Arguments: a constraint name which is of type string.
Synopsis: (gen-delete-constraint "Cnst-1")

Description: These functions delete/enable/disable a constraint.

Since the command Default-modes presents a menu and lets the user set the values of some options in the program, several functions are provided here, one for each option in this menu to set their values.

gen-set-step-mode :

gen-set-update-belief* :

gen-set-update-lambda-pis :

gen-set-compute-measures-of-impact :

gen-set-compile-data-file :

Optional Arguments: new value that this should be set to. This is of type boolean (t nil).

Synopsis: (gen-set-update-belief*)

Description: All these functions set that optional to the supplied value. If the argument is not supplied, then it toggles the current option. Note: if the argument is not supplied the system won't prompt for the argument. Instead it toggles the current option.

gen-set-data-pathname :

Optional Arguments: a pathname which is of type pathname.

Synopsis: (gen-set-data-pathname "local:>dir1:>dir2>") on the symbolics. Like wise (gen-set-data-pathname "usr/dir1/dir2/") on the sun.

Description: This function sets the current default data pathname to the given pathname.

gen-delete-model :

Optional Arguments: a new model name which is of type string. This name should be a valid model name.

Synopsis: (gen-delete-model "model-3")

Description: This function deletes the current model and then sets the current model to the supplied model name.

gen-add-evidence :

Optional Arguments: a valid node name and new evidence. Node name is of type string. New evidence is a list of numbers between 0.0 and 1.0 such that the sum of all these numbers equal to 1.0 and the total number of numbers should be equal to the cardinality(number of values that the node takes) of the node.

Synopsis: (gen-add-evidence "node-A" '(.25 .5 .25))

Description: This function adds the evidence to the node.

gen-delete-evidence :

Optional Arguments: a node name and an integer. The node name is of type string.

Synopsis: (gen-delete-evidence "node-A" 0)
Description: This function deletes the nth (second argument) evidence from the node. The integer as the second argument indicates which evidence to delete. The numbers are chronological. 1 is for the first evidence entered, 2 for the second evidence entered etc. 0 is to indicate all the evidences at this node.

gen-read-evidence :
Optional Arguments: a file name which is of type pathname.
Synopsis: (gen-read-evidence "evidences-file.lisp")
Description: This function reads the supplied file which contains evidences to nodes in the current network.

gen-write-evidence :
Optional Arguments: a file name which is of type pathname.
Synopsis: (gen-write-evidence "evidence-file.lisp")
Description: This function writes all the evidences entered so far for the current network into that file.

gen-load-fiie :
Optional Arguments: a file name which is of type pathname.
Synopsis: (gen-load-file "inputfile")
Description: This function loads the given data file. If that is not an internal file, it may create an internal file depending on the *compile-file* option.

gen-load-model :
Optional Arguments: a model name which is of type string.
Synopsis: (gen-load-model "model3")
Description: This function makes the given previously loaded model current. The model must be previously loaded.

gen-prior-probs :
Optional Arguments: a node name and its new prior probability. The node name is of type string, and the prior probability is a list of numbers between 0.0 and 1.0. The number of elements in the list should be equal to the cardinality of the given node and the sum of these numbers should be equal to 1.0.
Synopsis: (gen-prior-prob "node-C" '(.2 .3 .5))
Description: This function sets the prior probability of the given node to the new prior probability supplied. If the current network is a network with loops, it also finds new relationships in the auxiliary network. Reconfiguring the relationships (JCPs) in the auxiliary network may take some time.

gen-node-info :

Optional Arguments: a node name which is of type string.
Synopsis: (gen-node-info "node-C")
Description: This function presents information about the selected node and all the links that this node is attached to.

gen-add-query :

Optional Arguments: a query node name and an expression. Both the arguments are of type string.
Synopsis: (gen-add-query "Quer-nd-1" "(and (= node-A val1) (= node-B val3))")
Description: This function creates a query node satisfying the given expression and names it as the node name supplied. Note: It is an error to create a query node with an already existing node name.

gen-delete-query :

Optional Arguments: a query node name which is of type string.
Synopsis: (gen-delete-query "Quer-nd-1")
Description: This function deletes the query node supplied from the current network.

gen-reset :

Optional Arguments: none.
Synopsis: (gen-reset)
Description: This function resets the current network to its original equilibrium state.

gen-targetnode :

Optional Arguments: a node name of type string.
Synopsis: (gen-target-node "node-C")
Description: This function makes the supplied node the target node in the current network and finds the measures of impact of the rest of the nodes in the network with respect to this node.

gen-update :

Optional Arguments: none.
Synopsis: (gen-update)
Description: This function brings the current network into equilibrium.

Some of the window interface commands like *revert-net, Zoom* are not applicable in TTY mode.

Appendix C - BaRT Functions Which Can be Called from Another Program

BaRT functions can be called from other programs as long as the user is in a Common LISP environment with or without PCL. Load the files exactly as in *Appendix B*. The tty functions described in *Appendix B* can be used from other programs. These functions prompt the user for missing arguments and take those values first. Then they do some error checking and call the corresponding core functions defined in the package *bart*. The user can call these functions with all the arguments directly. This reduces some overhead and provides more control.

All the node/link/network names in BaRT have internal names (pointers). Some of these functions take an internal name as an argument. Three macros *get-ptr, get-ptr-i get-net* have been provided to get the internal node/link/network from the external name. To avoid confusion, we say internal name where an internal name should be passed as an argument. Unless explicitly stated that the argument is internal, all the names are external. All external names are of type string and case sensitive.

The inner core functions are described later in this section. All these internal functions return true unless the user calls them with an incorrect argument. In this case, a list of n elements containing error information is returned. The user can process this list however is most convenient for him. The first element in the list is the atom *&err&*. The second element is an integer which can be decoded as follows:

1 -- The given network <arg> is illegal
2 -- New evidence given <arg> is not a list
3 -- All the elements in the given new evidence <arg> are not numbers
4 -- Length of the new evidence supplied <arg1> is not equal to the rank of the node <arg2>
5 -- New evidence is not supplied for the node <arg>
6 -- Illegal node name <arg>
7 -- File <arg> doesn't exist
8 -- Initial equilibrium has not yet been reached for the net <arg>
9 -- Illegal list of node names <arg>
10 -- Illegal list of link names <arg>
11 -- Illegal link name <arg>
12 -- Illegal list of object names <arg>
13 -- Illegal object name <arg>
14 -- Illegal evidence node/link
15 -- node <arg> has more than 1 evidence
16 -- no evidence present for <arg> node
17 -- New evidence supplied is not a number
18 -- Given node is neither a belief node not a taxonomical node
19 -- No such evidence present at this node
20 -- Current network is not a belief network
21 -- Illegal arguments to the function.

22 -- 2nd argument passed to zoom is not a number
23 -- can't make a tax-net without components
24 -- Belief network subtype <arg> is none of nil, ordinary, clustering, stochastic
25 -- Belief network type <arg> is none of tax-net, bel-net, or id-net
26 - parents <arg> in the given expression are not defined
27 -- Illegal node value pairs <arg> in the given expression
28 -- Illegal expression <arg>.
29 -- Can't create node <arg>. A node by that name already exists,
which cannot be deleted.
30 -- Can't create node <arg>. A node by that name already exists. Delete that node first.
31 -- Illegal file name : <arg>
32 -- Illegal network names: <arg>

The third through (n - 1)th elements are objects related to the error message. The last (nth) element is a string which states the error message.

get-ptr : (macro)

Arguments: external name of a node or a link and an external name of a network. Both are of type string.

Synopsis: (get-ptr "Coma" "med-net")

Description: Returns the internal pointer of the given node or link in the given network. The second argument is optional. In the absence of the second argument, the network defaults to the current network. Note that this macro returns only the internal pointer or nil; it does not return an error message if the arguments are incorrect. The user needs to use this wherever an internal node/link name is required as an argument to the core functions.

get-ptr-i : (macro)

Arguments: an external name of a node or a link and an internal name of a network. The first one is of type string, and the second one is pointer.

Synopsis: (get-ptr-i "Coma" cur-net-i)

Description: Returns the internal pointer to the node or the link in the given network. If the internal network name is not given in the list of parameters, it always defaults to the current network. Note that this macro returns only the internal pointer or nil; it does not return an error message if the arguments are incorrect.

get-net

Arguments: an external name of a network which is of type string.

Synopsis: (get-net "network-name")

Description: Returns the internal pointer of the given network. Note that this

macro returns only the internal pointer or nil; it does not return an error message if the arguments are incorrect.

change-net :

Arguments: external name of a network of type string.
Synopsis: (change-net "alarm-net")
Description: Makes the given network current.

add-constraint

Arguments: a constraint name and an expression. Both are of type string.
Synopsis: (add-constraint "Constraint1" "(= Headaches True)")
Description: This function creates a constraint with the given name satisfying the given expression in the current network.

int-gen-delete

Arguments: an internal node name of type pointer and an optional argument of type integer. This optional argument is valid only when the given node is a chance node.
Synopsis: (int-gen-delete (get-ptr-i "Constraint1")) (int-gen-delete (get-ptr "Query1" "med-net")) (int-gen-delete (get-ptr-i "node1") 0)
Description: This function is used to delete constraint/query/evidence nodes from the network. In case of a query or constraint node, the second argument is not valid. To delete an evidence node, the first argument must be the name of the chance node where this evidence is attached to, and the second argument is an integer corresponding to which evidence to delete. This integer should be between 0 and the number of evidences that node has. If 0 is passed as the second argument, then the system deletes all the evidences attached to this node. If a valid number other than 0 is supplied then the system deletes that evidence from the node.

act-inact

Arguments: an internal name of a constraint node of type ptr, and an integer from the set {-1 0 1}.
Synopsis: (act-inact (get-ptr "Constraint1" "med-net") 1)
Description: This function is used to inactivate/toggle/activate the given constraint node depending on the value of the second argument -1/0/1 respectively.

delete-model

Arguments: a model name of type string.
Synopsis: (delete-model "model-2")
Description: This function deletes the given model and removes it from the system completely.

add-self

Arguments: an internal name of a node of type pointer, and an evidence vector of type list of numbers.

Synopsis: (add-self (get-ptr "Headaches" "med-net") '(1.0 0))

Description: This function adds the given evidence vector to the given node. The length of the evidence vector should be equal to the cardinality of the node. Each number in this vector should be between 0.0 and 1.0 and should add up to 1.0.

int-gen-write-evid-nodes

Arguments: an internal name of a network of type pointer and a file name of type pathname.

Synopsis: (int-gen-write-evid-nodes (get-net "med-net") "evid-info-file")

Description: This function writes all the evidences in the given network into the given filename. Note: the corresponding command to read evid nodes is not provided. Reading information about evidences is just to load the file containing the information about evidences in a network. Example: (load "evid-info-file").

int-gen-load

Arguments: a file name of type pathname.

Synopsis: (int-gen-load "yourbartinputfile")

Description: This function reads the given file containing model and network information suitable for bart to load and loads that file. Depending on the *compile-file* option, this may create an internal compiled file suitable for bart to load next time.

replace-prior

Arguments: an internal name of a node of type pointer, and a new prior probability vector of type list of numbers.

Synopsis: (replace-prior (get-ptr "Cancer" "med-net") '(.2 .8))

Description: This function replaces the prior probability of the given top node to the newly supplied prior probability vector. The length of this vector should be equal to the cardinality of the top node and each number in this list should be between 0.0 and 1.0. The total of all the numbers in this vector should add up to 1.0.

display-self

Arguments: an internal name of a node or a link of type pointer, and an output stream of type stream.

Synopsis: (display-self (get-ptr "Coma" "med-net") t)

Description: Displays information about the given node/link on the given output stream. This function takes two arguments: the internal name of a node or a link and a stream and displays the information about the first argument onto the stream.

add-query

Arguments: a query node name, and an expression. Both are of type sting.
Synopsis: (add-query "Query1" "(= Cancer True)")
Description: This function creates a query node satisfying the given expression.

int-gen-revert-net

Arguments: an internal name of a network of type pointer.
Synopsis: (int-gen-revert-net (get-net "med-net"))
Description: This function resets the given network to its initial equilibrium state. This removes all evidences attached and disables all the constraint nodes in that network.

int-gen-target-node

Arguments: an internal name of a node of type string.
Synopsis: (int-gen-targetnode (get-ptr "Cancer" "med-net"))
Description: This function makes the given node a target node in the current network and calculates all the benefit factors.

update-net

Arguments: an internal name of a network of type pointer.
Synopsis: (update-net (get-net "med-net"))
Description: This function brings the given network into equilibrium.

In addition to the above functions, the user has the ability to set some system options directly. But this is strongly discouraged. Instead the user should use the functions *gen-set-update-belief** etc. described in appendix B to set the system options.

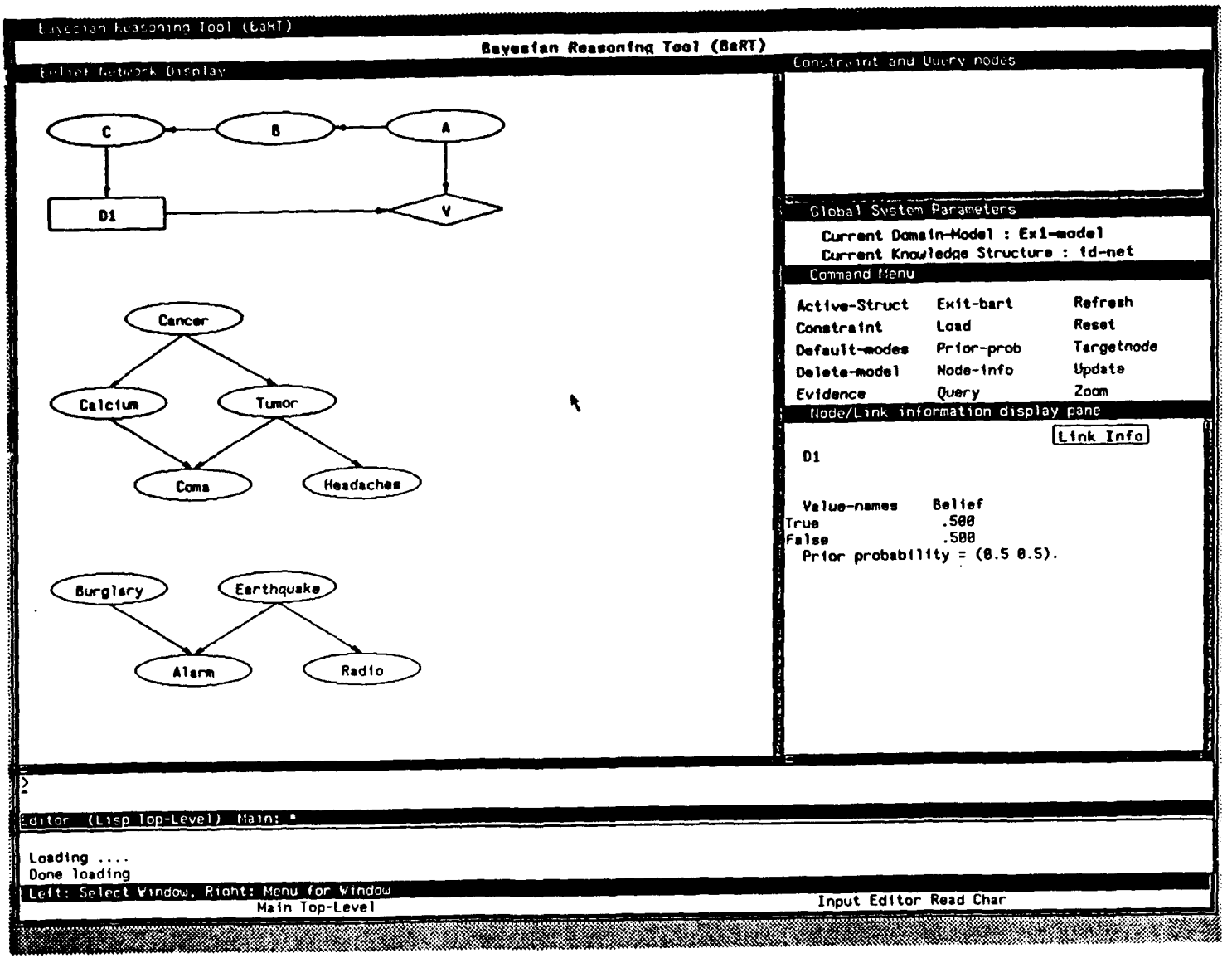
Bayesian Reasoning Tool (BaRT)
Belief Network Display
C
B
A
D1
V
Cancer
Calcium
Tumor
Coma
Headaches
Burglary
Earthquake
Alarm
Radio
Constraint and Query nodes
Global System Parameters
Current Domain-Model : Ex1-model
Current Knowledge Structure : id-net
Command Menu
Active-Struct
Exit-bart
Refresh
Constraint
Load
Reset
Default-modes
Prior-prob
Targetnode
Delete-model
Node-info
Update
Evidence
Query
Zoom
Node/Link information display pane
Link Info
D1
Value-names
Belief
True
.500
False
.500
Prior probability = (0.5 0.5).
Editor (Lisp Top-Level) Main: *
Loading ....
Done loading
Left: Select Window, Right: Menu for Window
Main Top-Level
Input Editor Read Char

Figure 1

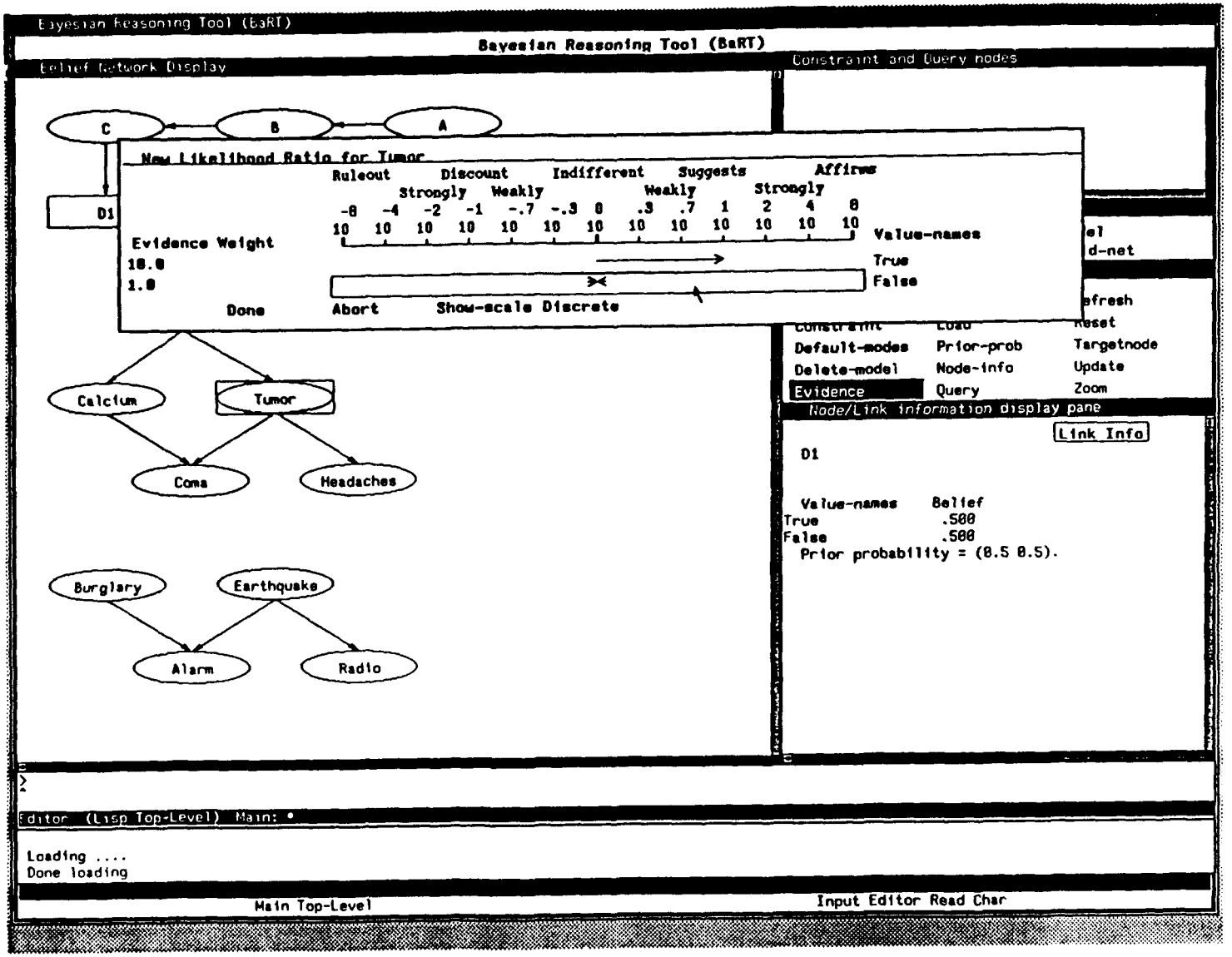
Bayesian Reasoning Tool (BaRT)
Bayesian Reasoning Tool (BaRT)
Belief Network Display
Constraint and Query nodes
C
B
A
D1
New Likelihood Ratio for Tumor
Ruleout
Discount
Indifferent
Suggests
Affirms
Strongly
Weakly
Weakly
Strongly
-8 -4 -2 -1 -.7 -.3 0 .3 .7 1 2 4 8
10 10 10 10 10 10 10 10 10 10 10 10 10
Evidence Weight
Value-names
True
False
18.0
1.0
Done
Abort
Show-scale Discrete
Calcium
Tumor
Coma
Headaches
Burglary
Earthquake
Alarm
Radio
Default-modes
Prior-prob
Delete-model
Node-info
Update
Evidence
Query
Zoom
Node/Link information display pane
Link Info
D1
Value-names
Belief
True
.500
False
.500
Prior probability = (0.5 0.5).
Editor (Lisp Top-Level) Main: *
Loading ....
Done loading
Main Top-Level
Input Editor Read Char

Figure 2

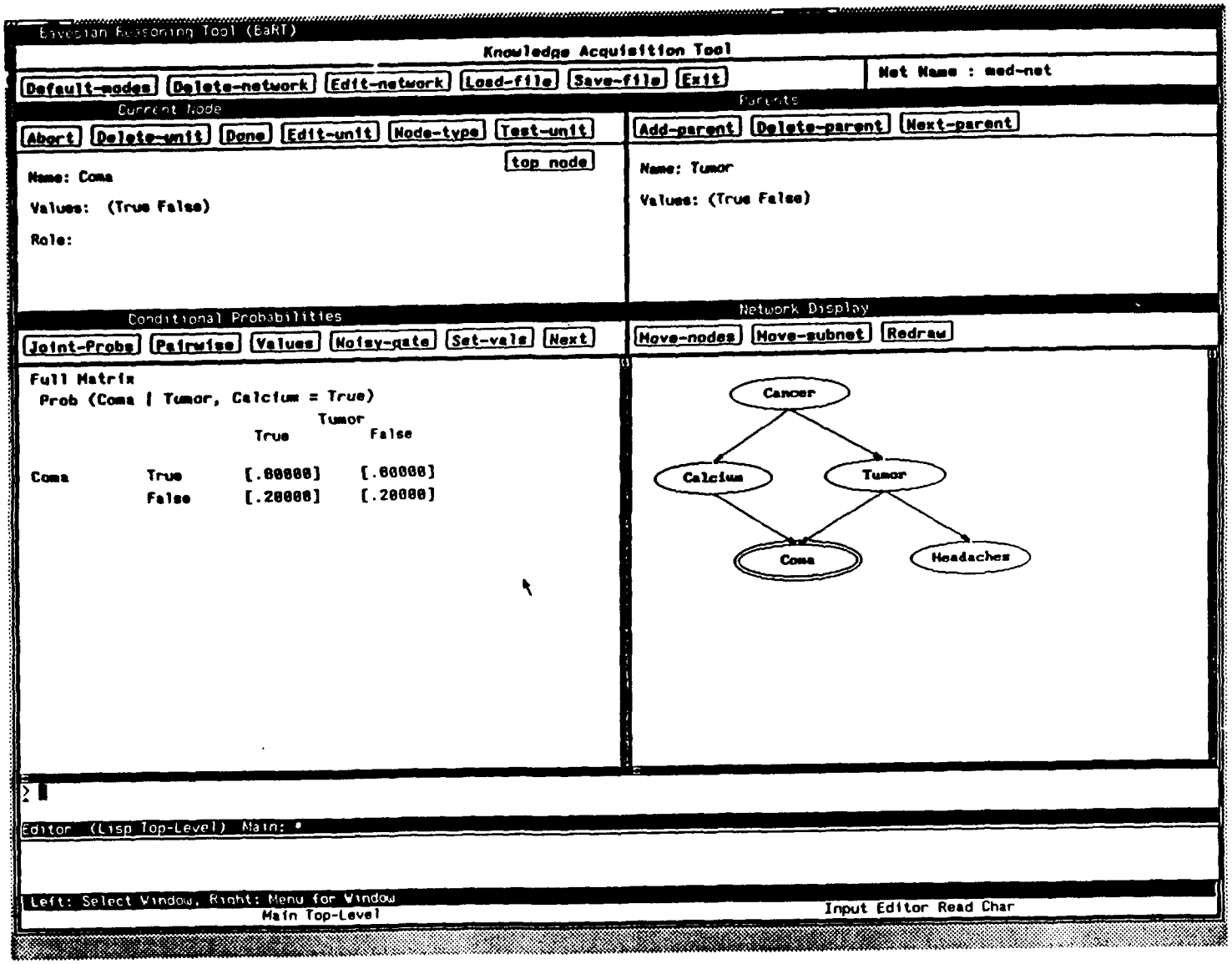
Bayesian Reasoning Tool (BaRT)
Knowledge Acquisition Tool
Default-nodes
Delete-network
Edit-network
Load-file
Save-file
Exit
Net Name : med-net
Current Node
Parents
Abort
Delete-unit
Done
Edit-unit
Node-type
Test-unit
top node
Add-parent
Delete-parent
Next-parent
Name: Coma
Values: (True False)
Role:
Name: Tumor
Values: (True False)
Conditional Probabilities
Network Display
Joint-Probs
Pairwise
Values
Noisy-gate
Set-vals
Next
Move-nodes
Move-subnet
Redraw
Full Matrix
Prob (Coma | Tumor, Calcium = True)
Tumor
True
False
Coma
True
[.80000]
[.60000]
False
[.20000]
[.20000]
Cancer
Calcium
Tumor
Coma
Headaches
Editor (Lisp Top-Level) Main: *
Left: Select Window, Right: Menu for Window
Main Top-Level
Input Editor Read Char

Figure 3